Unit title	Unit 4: Programming
Guided learning hours	90
Number of lessons	45
Duration of lessons	2 hours

Key to learning opportunities			
IA	Internal Assessment	**RS**	Revision session
GS	Guest speaker	**V**	Visit
IS	Independent study	**C**	Classroom activities directed by teacher

Lesson	Topic	Lesson type	Suggested activities	Resources
Learning aim A: Examine the computational thinking skills and principles of computer programming				
1	**A1 Computational thinking skills** Decomposition	IS C	• **Lead-in:** Introduce the concept of computational thinking (CT) and stages of CT using tutor presentation or extracts from Google CT website. Explain that CT will be applied throughout this and other units. • **Small-group or paired activity:** Give learners a scenario in which they are asked to consider the different problems that will need solving if they were to create a social media app. • **Tutor-led discussion:** Learners feed their thoughts back to the whole group. • **Plenary:** Using what they have learned from the discussion, learners complete work on the scenario activity.	• Flipchart or similar for learners to record discussions and ideas.
2	**A1 Computational thinking skills** Decomposition Pattern recognition Pattern generalisation and abstraction	IS C	• **Lead-in:** Recap the concept of decomposition with a Q&A session. Explain that in this lesson they will continue to explore CT. • **Small-group or paired activity:** In the same groups as the previous lesson, learners expand their analysis of the problem of creating a social media app (e.g. identify patterns and common elements or features).	• Example social media apps.

Lesson	Topic	Lesson type	Suggested activities	Resources
			Introduce some programming terms and ask learners to identify variables in such programs (e.g. 'character count' – for counting the number of characters in a status update). • **Tutor-led discussion:** Ask learners to contribute their thoughts to the discussion. • **Plenary:** Using what they have learned from the discussion, learners complete work on the scenario activity.	
3	**A1 Computational thinking skills** Decomposition Pattern recognition Pattern generalisation and abstraction Algorithm design	IS C	• **Lead-in:** Recap the concepts of 'decomposition', 'pattern recognition' and 'pattern generalisation and abstraction' through Q&A session. Explain that over the next three lessons learners will explore the first stages of algorithm design. • **Small-group or paired activity:** Give learners a scenario where they must reverse engineer the classic computer game 'pong'. In groups, learners should decompose the problem, identify repeated patterns, list the variables they may need etc. Next, learners consider the 'algorithms' for different parts of the game (e.g. controlling the bat, movement of the ball, scoring points). Highlight that in the early design stages of a	• Large sheets of paper. • Example game for learners to examine. • Different coloured pens. • (Optional) 'Scratch' or other similar simple code

Lesson	Topic	Lesson type	Suggested activities	Resources
			software solution the 'algorithms' only need to be step-by-step instructions, using everyday language, and do not need to be presented in pseudocode or full programming code. • **Group activity (peer review and feedback):** At different stages during the lessons, ask groups to share their work with other small groups. Each group should give feedback to each other about the algorithms produced. Feedback should focus on completeness and accuracy. • **Plenaries:** Supply groups with different coloured pens to help them annotate work when giving feedback, and so they can annotate changes they have made as a result of feedback. • **Independent learning activity:** If time and resources allow, give learners a chance to implement some of their algorithms into a working system, e.g. the 'controlling the bat' algorithm using Scratch. Learners could use pre-made sprites and apply their algorithm. They should check their algorithm by running the 'code' to see if it performs as they would expect.	development program.

Lesson	Topic	Lesson type	Suggested activities	Resources
4	**A2 Uses of software applications** Gaming and entertainment Productivity	IS C	• **Lead-in:** Introduce the lesson, drawing attention to the different uses of software applications to be covered over the next six lessons. • **Independent learning activity:** Learners analyse how different 'gaming and entertainment' applications and 'productivity' applications meet user needs (e.g. functions the applications perform, interaction with users/other programs, delivery platform etc). • **Small-group discussion:** In pairs or small groups, learners discuss their findings and share ideas.	• Computers with internet access for research. • Example gaming and entertainment applications. • Example productivity applications.
5	**A2 Uses of software applications** Information storage and management	IS C	• **Lead-in:** Introduce the lesson and clarify learners' understanding of 'problem-solving' through Q&A of areas covered over the last two lessons. • **Small-group activity:** In pairs or small groups, conduct research into the functions, features and implications of information storage and management software. • **Class discussion:** As a class draw together findings from the research. Tutor to record outcomes and share a copy with the learners.	• Computers with internet access for research.

Lesson	Topic	Lesson type	Suggested activities	Resources
6	**A2 Uses of software applications** Repetitive tasks or dangerous tasks		• **Lead-in:** Introduce the purpose of the lesson. Use Q&A to establish a baseline understanding of vocational areas that would require software for repetitive or dangerous tasks. • **Individual learning activity:** Supply learners with vocational scenarios that would require software to aid with repetitive and/or dangerous tasks. Learners should analyse the implications of using software to aid with these tasks. • **Class discussion:** As a class, draw together key observations. Tutor to record outcomes and share notes with the learners for future use.	• Computers with internet access for research. • Scenarios for tasks.
7	**A2 Uses of software applications** Social media Search Engines	IS C	• **Lead-in:** Introduce the purpose of the lesson: to analyse how social media apps meet user needs. • **Independent learning activity:** Learners analyse how different social media applications and search engines meet user needs (e.g. functions the applications perform, interaction with users/other programs, delivery platform etc). Learners should include consideration of some of the programming problems that are present in creating such a program (e.g.	• Computers with internet access for research. • Example social media applications. • Access to different

Lesson	Topic	Lesson type	Suggested activities	Resources
			drawing data from different sources, authentication/security etc). • **Small-group discussion:** In pairs or small groups, learners discuss their findings and share ideas. • **Plenary:** Using what they have learned from the discussion, learners expand their analysis notes.	search engines.
8	**A3 Features and characteristics of programming languages** The uses and applications of high-level programming languages Procedural Object Orientated	IS C	• **Lead-in:** Explain to learners that they will now look at specific programming paradigms in relation to specific languages. • **Tutor presentation:** Give an overview of the features and characteristics of the procedural and object orientated programming paradigms. • **Individual task:** Learners should research and make notes on the uses, benefits, and drawbacks of procedural and object orientated programming. • **Small-group task:** Reorganise learners into small groups to discuss their findings. Allow them to add to and expand their notes based on the discussions.	• Computers with internet access for research.

Lesson	Topic	Lesson type	Suggested activities	Resources
9	**A3 Features and characteristics of programming languages** The uses and applications of high-level programming languages Event-driven Mark-up	IS C	• **Lead-in:** Introduce the purpose of the lesson: to explore the uses and implications of the event-driven programming paradigm. • **Tutor presentation:** Give an overview of the features and characteristics of event-driven and mark-up programming and compare to procedural programming and OOP. • **Individual task:** Learners should research and make notes on the uses, benefits, and drawbacks of event-driven and mark-up programming. • **Small-group task:** Reorganise learners into small groups to discuss their findings. Allow them to add to and expand their notes based on the discussions.	• Computers with internet access for research.
10	**A4 Constructs and techniques and their implementation in different languages** Command words	IS C	• **Lead-in:** Give learners a simple algorithm, written in pseudocode and shown as a flow chart that describes part of a simple game (e.g. Pac-Man). Ask them to look at the algorithm and explain what they think it is for/what it does. • **Tutor presentation:** Introduce the concept of pseudocode when designing algorithms and planning computer programs. Explain why pseudocode is used	• Example algorithm and flow chart for starter. • List of basic operations.

Lesson	Topic	Lesson type	Suggested activities	Resources
	constants and variables, local and global variables data types – character, string, inte.g.er, real, Boolean statements – assignment, input and output, sequence, iteration, selection File handling Logical operations		as a planning and design tool. Introduce the 'basic operations' listed in the specification. Explain that operations, key commands and common programming functions are typically written in upper case within the pseudocode. • **Independent learning activity:** Give learners simple scenarios and flow charts or components of a computer program that they must present as pseudocode. Focus on simple command words and file handling commands avoiding, if possible, decisions and repetition. However, depending on learners' ability and their programming experience, you may decide to introduce concepts from later lessons. • **Plenary:** Ask learners to discuss/compare their solutions in small groups. Ask one group to share their solutions. As a class, discuss the merits of each solution (try to identify a group that has two varying but still valid solutions). Highlight that, depending on the scenarios, there may be different ways of presenting solutions which are still valid, just as there are when creating a programmed solution.	• Scenarios for pseudocode task.

Lesson	Topic	Lesson type	Suggested activities	Resources
11	**A4 Constructs and techniques and their implementation in different languages** String handling functions	IS C	• **Lead-in:** Explain that learners will continue to look at common built-in functions that are available in many high-level programming languages. • **Tutor presentation:** Introduce the common string handling functions. Explain their purpose and use, and how they may be used to add functionality to a program. See the links: Common string operations in Python, String operations in Java and C++ Strings in the resources column. • **Individual learning activity:** Give learners a series of problems involving string handling. Ensure problems include a wide range of contexts in which learners must apply understanding. Allow time to discuss solutions with individuals or as small groups. • **Plenary/assessment opportunity:** Give learners a problem that requires the development of code. Try to supply a problem that requires application of arithmetic functions and string handling functions. The task should be completed in exam conditions and marked by the tutor to gauge progress. Adjust the support you give depending on the learners.	• List of common string handling functions. • Example code. • Scenarios for tasks.

Lesson	Topic	Lesson type	Suggested activities	Resources
12	**A4 Constructs and techniques and their implementation in different languages** Arrays Data structures	IS C	• **Lead-in:** Supply a brief introduction to the concepts of data structures/abstract data types. Explain how the lesson will expand on previous lessons, which have focused on single value variables. • **Independent learning activity:** Learners do independent research into 'lists' and 'arrays'. They should research the characteristics of, and similarities and differences between, the two data structures. Learners should understand that different methods apply to different data types. See links: Array data type and Sorting animations in the resources column. • **Class discussion:** Use Q&A to check learners' understanding of lists and arrays. • **Independent task:** Give learners a series of scenarios/programming problems that require the use of data structures. They should identify in each case if a list or an array should be used and explain why. • **Small-group task:** Ask learners to form small groups to discuss their answers.	• Scenarios for tasks.

Lesson	Topic	Lesson type	Suggested activities	Resources
13	**A5 Principles of logic applied to program design** Iteration Mathematical logic Propositional dynamic logic	IS C	• **Lead-in:** Start with a practical activity. Give learners a spreadsheet or simple flat file database containing a series of different fields. Ensure some fields have validation and others do not. Only some validation should have appropriate error messages. Learners should enter data from the test plan or list, and record what happens. • **Class discussion:** Discuss what happened when the data was entered. Explore why only some fields behaved as expected. • **Tutor presentation:** Show learners the validation settings of the different fields. Show the different ways that entry is checked/restricted (give enough examples to cover a range of mathematical and logical techniques). Explain that although learners are currently looking at validation in a spreadsheet/database, the basic principles and logic are always the same. Explore the importance of validation within computer programs and systems that handle data. • **Individual task:** Give learners a series of common scenarios that might require data to be held. In each	• Spreadsheet/ database for starter. • List of test data/test plan. • Scenarios for tasks.

Lesson	Topic	Lesson type	Suggested activities	Resources
			case, give learners some fields for which they should develop validation rules. Learners do not need to implement the rules in a database or program; they could just give a clear description of what the validation would be. • **Plenary:** Ask learners to work in pairs to compare and contrast their solutions.	
14	**A5 Principles of logic applied to program design** Iteration Mathematical logic Propositional dynamic logic Sets	IS C	• **Lead-in:** Recap the last lesson • **Individual task:** Give learners a series of scenarios which require validation techniques to be applied. Learners should write/develop/improve code that could be applied to produce sections of a larger program. Learners may wish to plan this in pseudocode before implementing it in a programming language. • **Class discussion:** Discuss the solutions. Consider the different ways in which problems could have been solved. Explore the solutions to see if they can give robust validation. • **Plenary:** Further to discussion, allow learners to revisit solutions and refine their work.	• Scenarios for tasks.

Lesson	Topic	Lesson type	Suggested activities	Resources
15	**A6 Quality of software applications**	IS C	• **Lead-in:** Introduce the focus of the lesson: ways of checking the quality of software applications. • **Tutor presentation:** Explain how the quality of a piece of software can be measured in different ways by drawing learners' attention to the list of quality measures in the specification (topic A6). Introduce different tools and methodologies that can be used to check and measure quality. • **Individual or small-group activities:** Supply learners with examples of programs and code bases that they should check for quality. Learners should apply a range of different testing and quality assurance methods and report back on their findings.	• Example code for tasks. • Example software applications for tasks.
16-19	**Assignment 1** (learning aim A)	IA	• **Lead-in:** Introduce the assignment (see authorised assessment briefs) detailing the main assessment criteria, deadline, and submission criteria. • **Assignment writing:** Using the authorised assignment briefs, learners should complete the first assignment. They should explore how computational thinking skills and the principles of computer programming are used in a range of contexts and languages to create high	• Assessment brief.

Lesson	Topic	Lesson type	Suggested activities	Resources
			quality computer programs to solve problems and meet user needs.	
Learning aim B: Design a software solution to meet client requirements				
20	**B1 Software development life cycle**	IS C	• **Lead-in/tutor presentation:** Introduce the concept of the software development life cycle (SDLC). Explain that there are different models of life cycle all of which have benefits and drawbacks. Learners should be aware that, for the assessment they are not required to follow a specific model, but should show that they have followed a cyclical, iterative approach to the development of their software solution. • **Small-group activity:** Supply learners with an example project scenario. In a pair or small group, learners should plan out the tasks that they would carry out at each stage of the SDLC to ensure that the example project meets its aims. • **Class discussion:** As a class, discuss learner responses. • **Plenary:** In groups, learners to update/amend their responses based on the class discussion.	• Scenario for task.
21	**B2 Software solutions design**	IS C	• **Lead-in:** Explain that in topic B2, learners will explore the purpose and content of planning documentation.	Example planning documentation.

Lesson	Topic	Lesson type	Suggested activities	Resources
			• **Paired activity:** Supply each pair with example planning documentation. The documentation should be for a range of different programming solutions and may be in different layouts/formats but should cover the specification points listed in topic B2. Learners should analyse the examples to identify common content, examples of good practice etc. • **Class discussion:** As a group, share learners' findings and summarise the key criteria for high quality planning documentation.	
22	**B1 Software development life cycle** **B2 Software solutions design** Feedback from others to help refine alternative design ideas	IS C	• **Lead-in:** Explain that learners will now look at reviewing and refining planning documentation. For assignment 2 they will need to be able to gather feedback and select, use, and justify feedback methods. • **Tutor presentation:** Introduce the concept of gathering feedback by giving some examples/common methods. Also cover the need to identify and select the correct type and number of users. • **Individual learning activity:** Learners should conduct further research into methods of collecting and	• Computers with internet access, for research.

Lesson	Topic	Lesson type	Suggested activities	Resources
			collating feedback, including how they can be implemented, and the benefits and drawbacks of each. • **Plenary:** As a class, discuss learners' findings. Allow time for learners to add to their notes because of the discussion.	
23-24	**Mock assessment**	IA	• **Tutor presentation:** Introduce the mock assessment. • **Assignment writing:** Learners work on a mock assessment focused on learning aim B (Planning and design solution). • **Independent learning activity:** Learners complete the mock assessment (further work will be needed on this outside of the classroom).	Mock assessment.
Learning aim C: Develop a software solution to meet client requirements				
25–30	**C1 Software solutions development**	IS C	• Learners should spend these lessons practising implementing the programming constructs in a high-level programming language. • **Tutor presentation:** Explain key syntax, commands, and procedures as they are introduced and how understanding of the programming language progresses.	• Interactive development environment (IDE) – for coding • Syntax and command

Lesson	Topic	Lesson type	Suggested activities	Resources
			• **Individual learning activities:** Give learners opportunities to write small amounts of code to solve problems and explore features of the programming language. They should also be given larger code examples that they must debug and improve. • **Assignment preparation/mock assessments:** You should give learners opportunities to practise and apply skills and processes required for the second assignment (e.g. responding to scenarios, testing and refining extracts of code). It would be beneficial for learners for tasks to be placed in realistic, vocational scenarios.	reference guides. • Scenarios and sample data for tasks.
31–33	**C2 Testing software solutions** **C3 Improvement, refinement and optimisation of software applications**	IS C	• **Lead-in:** Use Q&A to remind learners of some of the concepts covered during lesson 22 (i.e. the need to gather feedback, and check and refine solutions). Expand on the concept of constant testing and feedback, and the idea that feedback/testing can take different forms. • **Tutor presentation(s):** Introduce the idea of different types of testing used for computing systems (e.g. testing the functionality of specific parts of a system, benchmarking performance, acceptance testing etc).	• Example computer programs. • Project criteria. • Task lists.

Lesson	Topic	Lesson type	Suggested activities	Resources
			Explain the role of test users and how they should be selected. • **Small-group activity:** Give learners example computer programs/code bases and related project specifications/success criteria. In small groups, learners should identify how they would go about testing the system to ensure it was fully fit for purpose (ie identifying how they would test different parts of the system and/or the system as a whole, selecting test users etc). • **Plenary:** Reorganise the learners so each group joins with another. Learners should discuss their ideas. • **Tutor presentation:** Explain the importance of ongoing testing and maintenance activities for software solutions. Cover the points listed in topics C2 and C3.	
34-35	**C4 Review of software solutions** **C5 Skills knowledge and behaviours**	IS C	• **Lead-in:** Explain the importance of reflecting on the outcomes of a project and, where appropriate, individual performance.	• Extracts from the specification. • Assessment criteria.

Lesson	Topic	Lesson type	Suggested activities	Resources

			• **Tutor presentation:** Explain the areas that should be considered when evaluating outcomes (and individual performance). These should include:
			  ○ how far the solution met the identified requirements
			  ○ the efficiency of the solution
			  ○ ease of use of the solution
			  ○ issues arising during testing and maintenance
			  ○ stability of the solution
			  ○ the potential update schedule
			  ○ how the solution could be improved further
			  ○ alternative solutions that could be implemented if the task was repeated.
			• Where appropriate, relate these areas to real-world requirements. Learners should write an evaluation of the work they produced and their own performance. Supply learners with extracts from the specification and assessment descriptors (as appropriate) to facilitate high quality focused evaluation.	

Lesson	Topic	Lesson type	Suggested activities	Resources
36-45	**Assignment 2** (learning aims B and C)	IA C	• **Lead-in:** Introduce the assignment (see authorised assessment briefs) detailing the main assessment criteria, deadline and submission criteria. • **Assignment writing:** Using the authorised assignment briefs, learners should complete assignment 2. The assignment should have enough scope for the learner to identify the computing needs of a client, plan and implement a computer program to meet the client's needs, and work with a client to plan, implement, test, review and refine the solution. The 'client', where possible, should be a real-world client with whom the learner can engage. The project might be simulated, in that it may not be a live project. Engaging with local employers to give a vocational setting would be invaluable. The project brief should have enough scope for learners to achieve at all ability levels.	• Assessment briefs. • Interactive development environment. • Appropriate hardware. • Client information.